Abstract

This essay describes a research program at the intersection of physical theory and computability. A new self-reference problem is posed which it is argued shows that for any actually testable physical theory there necessarily exists a physical system that cannot be tested, namely the hardware system that implements a computation of the theory's predictions. The argument proceeds in two stages, the first is to present a universal model for a physical implementation of a computation. In the second step we create a chain of models each of which describes the dynamics of the hardware underlying the previous model. It is argued that the solution to the Physical Self-Reference Problem is the limit of this sequence, but that the error at each step grows exponentially with the step index. The result is that finite iterations do not converge to solution but rather diverge from it.

1 Introduction

My research has been focused on the intersection of computability theory and physical theory, and whether it is possible to have a 'Theory of Everything' (TOE) to which *all* questions can in principle be referred. Computability theory would suggest that the answer to this question is that no such theory is possible. However the dominant philosophical perspective in the physical sciences, particularly Physics and Neuroscience, seems to be the diametrically opposite perspective, that computability theory has nothing to say in these subject areas, and that ultimately a theory will exist that will be capable of anwering any question in these areas. In an attempt to address the arguments used by physical scientists, I have constructed what I call the Physical Self-Reference Problem.

The Physical Self-Reference Problem (PSRP) asks if it is possible to have a computation, or analytic expression, that describes the dynamics of the physical system that performed that computation, or generated the analytic expression. The key point being that in order to test a theory one needs to perform a computation in order to generate data to test. If the dynamics of that computation process necessarily exceed our modeling ability then there necessarily exists at least one physical system which we cannot model, and hence no 'Theory of Everything' is possible.

My work has focused primarily on the Turing computability case. Of the many other choices for what a calculation might look like, the analytic case seems to be the major alternative in actual use. The analytic case however has elements that have no clear physical implementation, in particular the taking of limits. The proof route that I have been pursuing consists of two main steps. The first is to establish a universal description of what a computation is, and more importantly a generic description of all possible dynamical implementations of a computation. The second step, which is the PSRP proper, consists of showing that the PSRP leads to an infinite chain of dynamical systems, each of greater complexity than the last.

2 Step One: The Building Blocks

In this section we describe the major entities needed to construct the Physical Self-Reference Problem. The main components are a computational model, and a hopefully universal description of any physical implementation of that computational model.

2.1 Computational Machinery

In order to facilitate later steps it is necessary to modify the standard Turing Machine (TM) in two ways. The first is to make it into a Mealy Machine. This is done in two parts. The first is to require transition outputs to be labelled by both a new symbol, *and* a 'move' where a move of 0 is allowed. In other words transitions are labelled by (σ:(σ',k)) where σ, $\sigma' \in \Sigma$ are elements of the alphabet of symbols and k$\in \mathbb{Z}$ indicates the next cell to act on. These modifications are a superset of the standard TM model and thus have equal computational power. Conceptually instead of thinking of a boxcar on rails we are thinking of a collection of variables connected by a switching network to a collection of operators $\{M_j | \, M_j : \Sigma \to \Sigma\}$ where j$\in J_{op} \subsetneq \mathbb{N}$ (J_{op} finite). (see figure 1 p. 2)

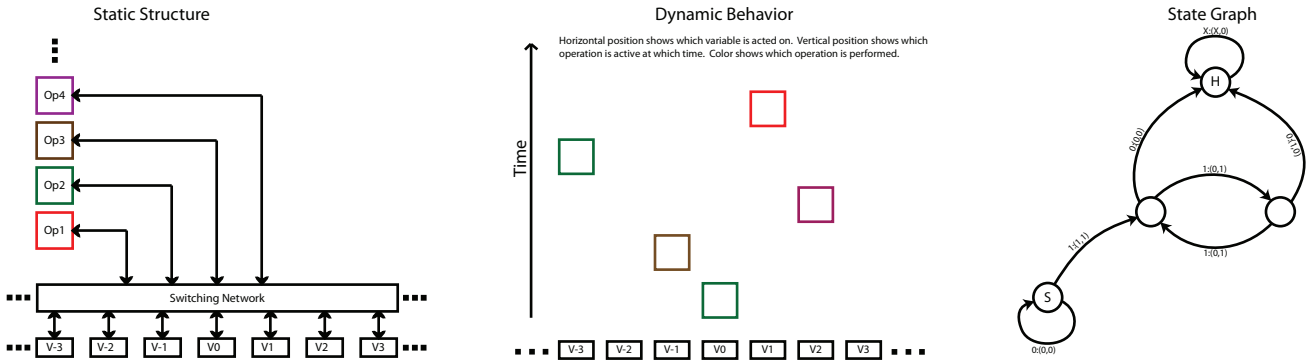

Figure 1: Three views of the Turing Machine as a switched network with a Mealy Machine state graph

The two outputs of a transition indicate 1) the operator to employ on the current variable, and 2) the next variable to act on. The second step consists of requiring that all nodes of the state machine have outgoing labels for every possible $\sigma \in \Sigma$. This change too has no effect on functionality and has the effect of making the modified TM into a well defined discrete time dynamical system, given by the following equation:

$$\vec{v}(t_H) = \begin{pmatrix} \vdots & & \vdots & \vdots & \vdots & \vdots \\ \cdots & M_{p(-1,\vec{v}(t),t_{i+1})} & 0 & 0 & \cdots \\ \cdots & 0 & M_{p(0,\vec{v}(t),t_{i+1})} & 0 & \cdots \\ \cdots & 0 & 0 & M_{p(1,\vec{v}(t),t_{i+1})} & \cdots \\ \vdots & & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} \vdots \\ v_{-1}(t_i) \\ v_0(t_i) \\ v_1(t_i) \\ \vdots \end{pmatrix} \tag{1}$$

where we are showing the description for a single step from time t_i to time t_{i+1}. The complete description is then a product of matrices of the above form where the index i runs from 0, the start time, to H $\in \mathbb{N}$ the halt time. The subscript $p(k, \vec{v}(t), t_i) : \mathbb{Z} \times \vec{v}(t) \times \mathbb{N} \rightarrow J_{op}$ indicates the actual path traversed in the graph of the TM and provides the specific endomorphism M_p that is applied to variable k (v_k) at time t_i. The $\vec{v}(t)$ dependence is shorthand indicating that the path depends on the prior data, as a result *generically* the TM is a non-linear system. We observe that the matrix is diagonal in the basis we are using. Notice that for any single time step all but one of the M_p's is an identity operator. We thus recover a result from the work of Blum, Shub and Smale on the BSS Machine[1], that the input/output map of any computation is a polynomial over each input/output pair. In fact in this case it is a monomial of operators acting on the initial variable value or input, and it follows that in this basis the equation representing the entire computation, like that for a single time step, is diagonal.

It turns out that there is a class of groups called 'Self-Similar Groups'[3] which are groups generated by Mealy Machines. It follows that with the above computational model, any computable function generates a self-similar group. The monograph *Self-Similar Groups*[3] provides an algebraic formalism (extending that in Eilenberg[2]) for these groups in terms of the state graph of the Mealy Machine. In particular the states Q are a bi-module over $\mathsf{G_s}$ where the group $\mathsf{G_s}$ can naturally be interpreted as a subgroup of the automorphism group $\mathrm{Aut}(\Sigma^*)$ of the rooted tree Σ^* extendable to $\mathrm{Aut}(\Sigma^\omega)$. The states Q can be identified with the direct product $\Sigma \times \mathsf{G_s}$ of the symbol alphabet and the group. The left action is given by $h \cdot (\sigma, g) = (h(\sigma), h|_\sigma\, g)$ where h,$g \in \mathsf{G_s}$, and $\sigma \in \Sigma$. Here $h(\sigma) = \sigma'$ which is the action of some endomorphism M_j specified by the transition between two nodes of the state graph. Similarly $h|_\sigma$ is the next state function, denoted schematically by the line that connects two states in the state graph. The right action of the bi-module Q is given by $(\sigma, g) \cdot h = (\sigma, gh)$. In other words, given a word $w \in \Sigma^*$ (or Σ^ω), we have $g \cdot \sigma = \sigma' \cdot h$ if and only if $g(\sigma w) = \sigma' h(w)$, and $g|_\sigma = h$ [3]. This means that the output letter, and the next state are encoded by the action of the input symbol on the bi-module Q. The module Q can, as we will see in the next section, be identified with the orbits of the physical system giving rise to the computation under the action of the time evolution operator.

2.2 Dynamical Machinery

In this section we describe the dynamical system associated to the computational system discussed in the previous section. Ideally there would be some universal prescription for converting an algebraic specification into a dynamical system, unfortunately while there is a minimal dynamical system we can naturally associate to such a description[4], there does not appear to be a natural way to define a *stable* dynamical system. Therefore we have to go through a tediously large range of cases, which we will only summarize in brief here. The overall goal here is to develop a minimal specification for a dynamical system in terms of number of zeros and in terms of number of functions needed

(minimally, in principle) to implement the computation.

To place the following discussion we first present a big-picture view of what we are going to do. We are going to first define the topology of a single variable, in as much generality as possible, and then define the other components we need, connectors and operators (see discussion of hardware implementation above) in terms of the topology of the variables. From these components we can thus build up a dynamical system that implements a particular computation.

2.2.1 Variables

The primary classification of variables from the point of view of the PSRP is into the categories of monolithic and composite coded variables. The key issue that divides them is how many *physical* functions are needed to build up one symbol of the alphabet. In monolithic coding only a single function is used (in principle), while in composite coding there exists a sub-symbol alphabet with each sub-symbol implemented minimally with a single function. Thus for composite codes $N_f > 1$, while for monolithic coding $N_f = 1$, where N_f is the number of functions needed to implement a variable. Within each of these categories we then have a number of sub-categories: space coded, function coded, and mixed space/function coded. The case of function coded also has within it a large set of possibilities: amplitude coding, frequency coding, phase coding, or as for instance in the case of the brain something even more complicated. The following illustration shows the major categories of hardware implementations of variables. (see figure 2 p. 4)

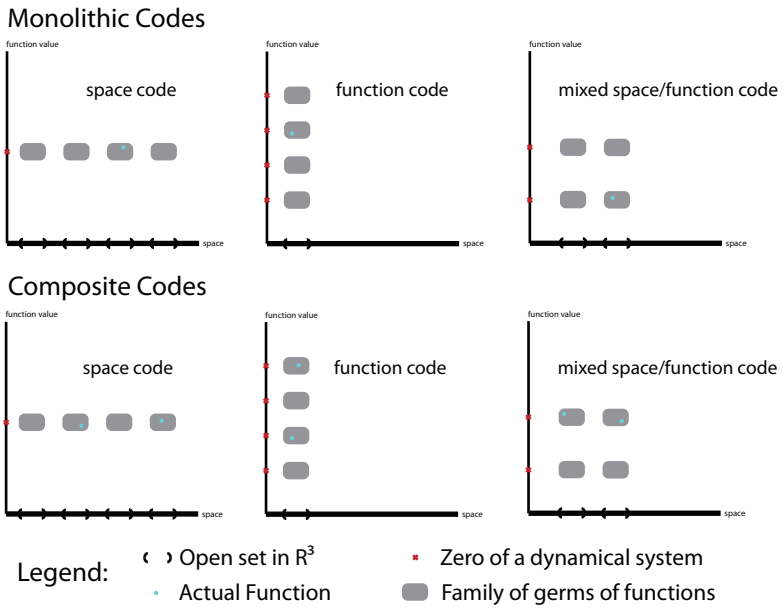

Figure 2: Monolithic vs. composite coding schemes. Note that all of the composite codes have multiple actual functions per letter while all of the monolithic codes have only one. The general classes of code types are represented in both the monolithic and composite cases, but we show only amplitude coding for the function coding case

It is possible to describe all of these case in a single mathematical description. A dynamical system $\mathscr{V}$ which implements a variable V contains a collection $\{\mathcal{V}_\sigma\}$ of open sets $\mathcal{V}_\sigma \subsetneq (\mathcal{U}, \mathcal{F})$ in the product space $\mathcal{U} \times \mathcal{F} \subset \mathscr{V}$, where $\mathcal{U} = \{U_i \subset \mathbb{R}^3 | \, U_i$ simply connected, open, $U_i \cap U_j = \phi$, and $\bigcup_{i \in I} U_i \subsetneq \mathbb{R}^3\}$, ($i \in I$ a finite index set $|I| \geq 1$), is an open set in physical space, and $\mathcal{F} = \bigsqcup_{i \in I} \mathcal{F}_i$, is the disjoint union of the rings of continuous functions, $\mathcal{F}_i$, on each of the components U_i of $\mathcal{U}$. Intuitively what we are saying overall is that there are a collection of open sets in space and on those open sets functions sufficiently close to an attractor map to a specific variable value σ, or more rigorously, there exists a map $\mathscr{G}$: $(\mathcal{U}, \mathcal{F}) \to \Sigma$ from a variable to the symbol alphabet, such that $(x, f(x)) \in \mathcal{V}_\sigma \Leftrightarrow \mathscr{G}(x, f(x)) = \sigma$. $\mathscr{G}$ is a union of constant maps whose domain is $\mathrm{Dom}(\mathscr{G}) = \cup_{\sigma \in \Sigma} \mathcal{V}_\sigma \subsetneq (\mathcal{U}, \mathcal{F})$.

For monolithic coding the number of attractors N_A equals the size of the alphabet $N_A = N_\Sigma$. For a composite code, let the subsymbol alphabet be Γ, and the number of active functions be N_f, then the size of the alphabet $N_\Sigma = |\Gamma|^{N_f}$. The number of attractors for a composite code is $N_A = |\Gamma| N_f = \frac{N_f N_\Sigma}{|\Gamma|^{N_f - 1}}$.

2.2.2 Connectors and Operators

The basic requirement that we have for a connector is that it preserve the mapping of the variable – that is that it take an equivalence class of functions at the start of the connector to the same equivalence class at the end of the connector. Clearly the easiest, but not the only, way to do this is to preserve the equivalence class throughout the connector. Furthermore we assume that it occupies some physical space and thus consists of a collection of connected open sets. The simplest case is where the number of components of a connector is equal to the number of components of a variable.

The operators too are defined on a number of connected components equal to the number of components of the variables. Any valid operator χ in the dynamical system $\mathscr{V}$ has to have the property that it takes a letter σ to some other letter σ', so $\chi_{\sigma,\sigma'} : (\mathcal{U}, \mathcal{F}) \to (\mathcal{U}, \mathcal{F})$ such that $\mathcal{V}_{\sigma'} \supseteq \chi_{\sigma,\sigma'} \mathcal{V}_\sigma$. In words, the map $\chi_{\sigma,\sigma'}$ has to be contracting on equivalence classes. For any symbol transformation there is an entire equivalence class of physical operators effecting the desired transformation, and any actual operator is one member of this equivalence class. The transformations associated with each of the variable types is shown below (figure 3 p. 6). The most generic statement about the form of the operators is that they are vector fields on braids in 3-space. In the case of composite coding a secondary coupling exists in order to move information from one sub-symbol to another.

At this point we have developed a formalism for describing the hardware underlying a Turing Machine. It is clear that this formalism implies the existence of some differential equation which can describe the time evolution of the system in question. However, in order to write down an actual equation we would have to choose a specific form for the variables, and their coding, which would limit the universality of the equation and the ensuing discussion. However we can give a

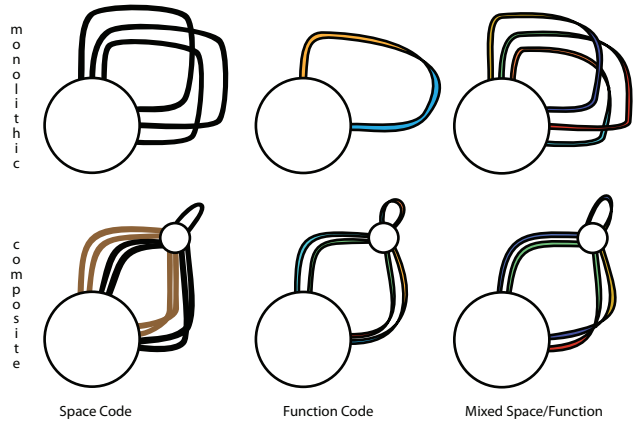

Figure 3: Illustration of the transformation effected by an operator for each coding scheme

graphical representation of it. (see figure 4 p. 6)

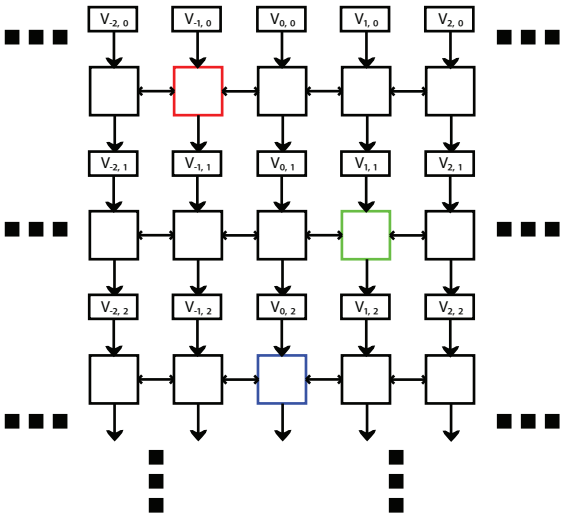

Figure 4: Turing Machine in implicit time form. In this picture each of the square boxes represents a member of an equivalence class of endomorphisms of the physical substrate of a variable. Black boxes are identity endomorphisms, the other colors represent non-identity endomorphisms.

The main point is that this is not a symmetric space – different locations in the space are different either in terms of the topology (pure space), vector fields on the topology (pure function), or both (mixed). So the equation is in fact a partial differential equation. This seems to imply that all of the foregoing discussion of actions of discrete groups on this space is best understood in the language of orbifolds, and Lie groupoids. We see also that generically the associated differential equation is non-linear – the operations applied depend on the initial data. What stands out clearly in the above picture is the presence of the group G_s – we can see how the state machine is expressed in the differential equation, and embodies the essential dynamics – and we recall that G_s incorporates the

non-linear data dependence of the equation. It turns out that the word structure of the boundary of the rooted tree Σ^ω generated by the group $\mathsf{G_s}$ is naturally a groupoid[6]. This appears to be a different groupoid than the groupoid of the operators of the differential equation, though they may be related.

3 Step Two: The PSRP Recursion

Having developed a description of both the behavior and implementation of a generic computation, we now consider the Physical Self Reference Problem itself. Throughout this section in the interests of brevity we describe only the major points with a minimum of motivation and details.

3.1 Seed Problem

The PSRP requires a seed problem. Any non-linear problem will do, so we focus on a non-linear problem having 2 zeros. We have in mind a physical system that gives rise to behavior similar to that of a simple binary symbol, and we call this system $\mathcal{D}_v$ for "Dynamical system of a Variable".

We define two equivalence classes of function values $\mathcal{V}_{0,\tau}$, and $\mathcal{V}_{1,\tau}$ by nearness to one or the other of the zeros at some specific time τ: $\mathcal{V}_{i,\tau} = \{(V \subsetneq \mathbb{R}^3, v_i = (l_i, u_i) \subsetneq \mathbb{R}) |\ y \in v_i \Rightarrow |y - \mathcal{Z}_i| \leq \epsilon\}$ for some $\epsilon \in \mathbb{R}$, i=1,2, where $y = f(x, \tau)$ and $f(x, t) \in H^0(\mathcal{D}_v, \mathbb{R})$ (the set V is the same in both of the $\mathcal{V}_{i,\tau}$). By making the size of the two open sets $\mathcal{V}_{i,\tau}$ very much larger than the size of the complement of their union, $\|\mathcal{V}_{0,\tau} \cup \mathcal{V}_{1,\tau}\| \gg \|\mathcal{V}_{0,\tau}^c \cap \mathcal{V}_{1,\tau}^c\|$, then the system will spend most of its time in one or the other of the $\mathcal{V}_{i,\tau}$ and the syntactic description of the variable in figure 5 (p. 7) will be coarse, but correct to a high degree of approximation. As a result of these manipulations we can associate a graph V_{ps} with the dynamical system $\mathcal{D}_v$:

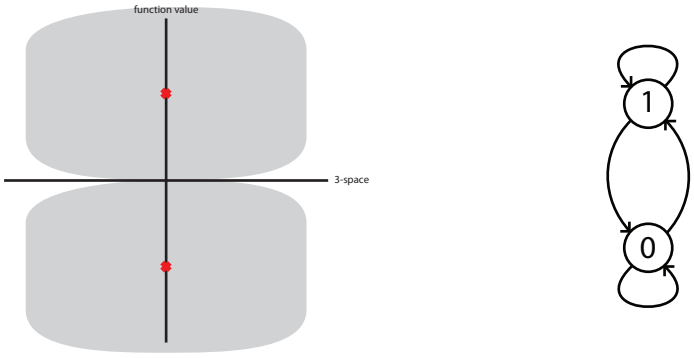

Figure 5: States of a binary variable. On the left we show the dynamical system $\mathcal{D}_v$, including the neighborhoods of the zeros used to define the syntactic structure. On the right we have the resulting discrete model

Evidently we can ask for a model M_1 of the system $\mathcal{D}_v$. Hence we have the following picture relating the three objects we have so far:

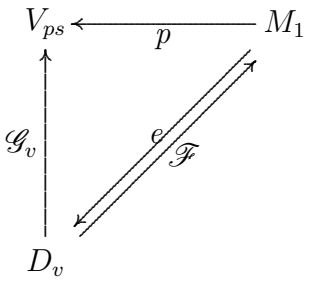

Figure 6: Initial segment of the PSRP model chain

The map $\mathscr{G}_v : \mathcal{D}_v \rightarrow V_{ps}$ is precisely the map we had in mind in defining the $\mathcal{V}_{i,\tau}$. It assigns an equivalence class of function values $\mathcal{V}_{i,\tau}$ (i=1,2) to a function value at some specified time τ: $\mathscr{G}_v(f,\tau) = \mathcal{V}_{v,i,\tau} \Leftrightarrow |f(x,\tau) - \mathcal{Z}_i| \leq \epsilon$.

The map e:$(\mathbb{N},\Sigma,\Sigma,\Sigma,\Sigma) \rightarrow (\mathcal{D}_v,\mathbb{R},\mathbb{R},\mathbb{R},\mathbb{R})$ takes a variable index $i \in \mathbb{N}$ and a quadruple of variable values $\sigma_0,\sigma_1,\sigma_2,\sigma_3 \in \Sigma$ to a point $p_i \in \mathcal{D}_v$ and a quadruple $(u_0,u_1,u_2,u_3) \in \mathbb{R}^4$, and we know that a family of first order function germs whose representative is given by $f(p \in V_{i,\sigma_0,\sigma_1,\sigma_2,\sigma_3}) = u_0 + u_1x + u_2y + u_3z$ defined on an open set $V_{i,\sigma_0,\sigma_1,\sigma_2,\sigma_3} = (U_i,(u_0-l,u_0+l),(u_1-l,u_1+l),(u_2-l,u_2+l),(u_3-l,u_3+l))$ surrounding (p_i,u_0,u_1,u_2,u_3), where $e(\mathrm{i},\sigma_0,\sigma_1,\sigma_2,\sigma_3) = (p_i,u_0,u_1,u_2,u_3)$, and $l = \frac{1}{2N_\Sigma}$. In other words $e = (\mathscr{F})^{-1}$ is the inverse of a map $\mathscr{F}$ (figure 7, p. 9) from a set of families $\mathcal{F}_{V_{i,\sigma_0,\sigma_1,\sigma_2,\sigma_3}}$ of germs of functions on $\mathcal{D}_v$, to the symbols in $\mathsf{M}_{1,sp}$ indexed by i. (see figure 7 p. 9)

In plainer language the values of a quadruple of variables tells us the value of the function and its first derivatives with some uncertainty determined by the size of the symbol alphabet and the number of symbols used to encode the complete function space of the manifold. In view of the inverse relationship between the map e and the map $\mathscr{F}$ we will from now on refer to the map e simply as $\mathscr{F}^{-1}$.

The map $p : \mathsf{M}_{1,ps} \rightarrow V_{ps}$ from the model to the graph is a simple projection. In brief for variables of M_1 that code for function values (not derivatives), when the value associated with the variable is in one of the sets $\mathcal{V}_{i,\tau}$ then p maps that *syntactic* 'value' (say σ) to the appropriate $\mathcal{V}_{i,\tau}$. In other words $p(v_k,\sigma) = \mathcal{V}_i \Leftrightarrow |f(q) - \mathcal{Z}_i| \leq \epsilon$ where $f(q) = \mathscr{F}^{-1}(k,\sigma,\Sigma,\Sigma,\Sigma)$ is the value of the function at the point q given by the map $\mathscr{F}^{-1}$. It follows that the set of maps shown in the diagram (figure 6 p. 8) form a commutative diagram, at least on the restriction of $\mathscr{F}$ to $\mathrm{Dom}(p)$ the domain of p. That is $\mathscr{G}_v = p \circ \mathscr{F}|_{Dom(p)}$.

3.2 The PSRP Recursion

Let us now consider the model M_1 independently of the variable it is a model of. Evidently if M_1 is to exist it must also be a physical system, and from the foregoing, we can look for a minimal

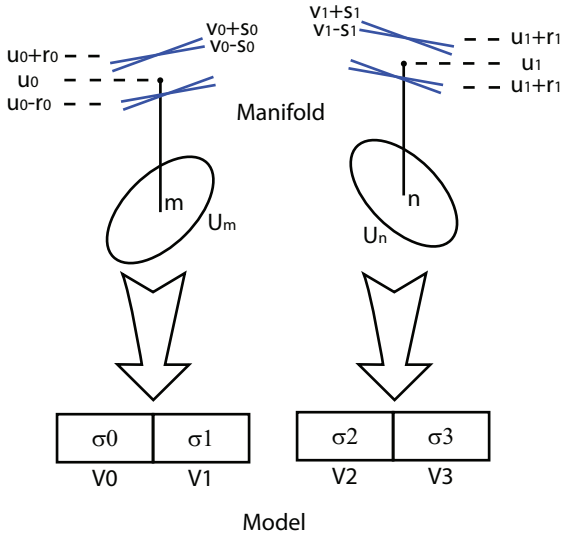

Figure 7: The map $\mathscr{F}$ from a manifold to its model. For simplicity we are only showing a single derivative in this figure

implementation of it. We thus get the following diagram:

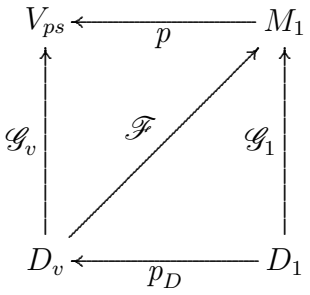

Figure 8: First step of the PSRP recursion

Where $\mathcal{D}_1$ is the dynamical system that implements or creates the symbol system M_1. Evidently $\mathcal{D}_1$ is a 'bigger' dynamical system than $\mathcal{D}_v$. Where $\mathcal{D}_v$ had precisely two zeros, $\mathcal{D}_1$ has a number of zeros strictly greater than two. Generically the variables of M_1 are elements in a very large alphabet, say $\Sigma_{\mathsf{M}_1} = \mathbb{Z}/(2^{32})\mathbb{Z}$, so even in a composite coding scheme with a subsymbol alphabet of $\mathbb{Z}/2\mathbb{Z}$ the number of zeros per variable is 64, a factor of 32 greater than the number of zeros in $\mathcal{D}_v$, before multiplication by the number of variables. Nor does this exhaust the set of zeros of M_1, because the vector fields implementing the endomorphisms also have zeros for function coding, or holes in the topology in the case of space coding.

The map p_D takes quadruples of zero-th order function germs in $\mathcal{D}_1$(physical instances of variables in M_1) into first order function germs in $\mathcal{D}_v$: $p_D : \{ (\mathcal{U}_{v_j}, \mathcal{F}_{v_j}) \}$ (j=0,1,2,3) $\rightarrow$ $(\mathcal{D}_v, \mathbb{R}^4)$ is given by $p_D(\mathcal{V}_{\sigma_0}, \mathcal{V}_{\sigma_1}, \mathcal{V}_{\sigma_2}, \mathcal{V}_{\sigma_3}) = (U_i, f)$ where the representative $f = u_0 + u_1 x + u_2 y + u_3 z$ is

given by the map $\mathscr{F}^{-1}(i,\, \sigma_1,\, \sigma_2,\, \sigma_3,\, \sigma_4) = (\, U_i,\, u_0,\, u_1,\, u_2,\, u_3)$ which also specifies the open set U_i. Note that p_D is bijective but is only defined on the equivalence classes $\mathcal{V}_{\sigma_j}$. Furthermore p_D^{-1} is an *expanding* map. It takes any member of a family of first order function germs in $\mathcal{D}_v$ to four topologically distinct components of $\mathcal{D}_1$. In addition we have that homotopy equivalent function germs in $\mathcal{D}_v$ become topologically separated in $\mathcal{D}_1$. We note also two commutative diagrams: $p_D = \mathcal{F}^{-1} \circ \mathscr{G}_1$, and $p_D|_{Dom(\mathscr{G}_v)} \circ \mathscr{G}_v = \mathscr{G}_1|_{Dom(p)} \circ p$. Finally we note that for the case of composite coding we have a distinct pair $(\mathcal{U}_{v_j},\, \mathcal{F}_{v_j})$ for each *sub-symbol*. Therefore in this case the map $p_D : \{\, (\mathcal{U}_{v_j},\, \mathcal{F}_{v_j})\,\}(j = 0, 1, 2, \cdots, 4N_f - 1) \rightarrow (\mathcal{D}_v, \mathbb{R}^4)$ takes one first order function germ of $\mathcal{D}_v$ into $4N_f$ topological components of $\mathcal{D}_1$.

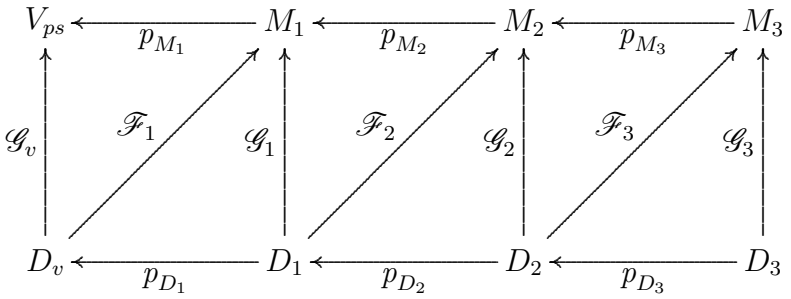

Figure 9: Second step of the PSRP recursion. Each model M_j is a model of the dynamical system $\mathcal{D}_{j-1}$. Each M_j is implemented by a dynamical system $\mathcal{D}_j$ which is more complex than the dynamical system $\mathcal{D}_{j-1}$. Thus no M_j is a model of itself. The solution to the physical self reference problem is thus the profinite completion of this sequence of models.

We notice that in figure 8 p. 9, V_{ps} is only a lowest order model of $\mathcal{D}_v$, and similarly M_1 is only a lowest order model of $\mathcal{D}_1$. Thus we need to solve for the behavior of $\mathcal{D}_1$. Clearly this requires introducing a new model M_2, but M_2 will then have a dynamical system that implements it, and this new dynamical system $\mathcal{D}_2$ will be more complex than $\mathcal{D}_1$, and its behavior will be effectively unknown, except to lowest order by M_2. Therefore a new model will be needed, and so on. (see figure 9 p. 10.) None of the models M_i in this chain are capable of describing syntactically the behavior of the functions on the dynamical system $\mathcal{D}_i$ giving rise to M_i (again except to lowest order), therefore none of these models is a solution of the PSRP. It seems reasonable to conjecture that the forward limit of the models M_i, and the corresponding forward limit of the dynamical sytems $\mathcal{D}_i$ would be able to solve the problem, however there are significant obstacles to achieving this limit.

We have shown previously that the number of zeros $N_{z,j}$ of $\mathcal{D}_j$ is strictly greater than the numbers of zeros $N_{z,j-1}$ of $\mathcal{D}_{j-1}$, indeed, $N_{z,j} = \lambda N_{z,j-1}$ for some $\lambda \in \mathbb{N}$ where $\lambda > 1$. Thus $N_{z,j} \geq \lambda^{j-1} N_{z,1}$ and $N_{z,\infty} = \lim_{j \to \infty} N_{z,j} = \infty$. Assume for the moment that we are willing to regard a function as

'well approximated' if we have N_p points at which the function value, and the first derivates are known (this constant number of terms for an entire function seems to undercount the number really needed). Further, let us denote by $N_{V,i}$ the number of variables needed in model M_i, and by $N_{f,i}$ the number of functions in $\mathcal{D}_i$. After a little calculation, one can show that if we stop at any finite stage j the number of symbols we need is $N_{V,j+1} = 4^{j+1}N_p^{j+1}N_f^j$. On the other hand the number of symbols we have is $N_{V,j} = 4^j N_p^j N_f^{j-1}$, so the difference between what we don't know, and what we do is $N_{V,j+1} - N_{V,j} = 4^j N_p^j N_f^{j-1}(4N_pN_f - 1)$. We have exponential growth in the error term. Thus for any finite approximation there is no convergence. This error estimate seems to be a coarse lower bound. The reason is that the requirement of a constant number of terms implicitly assumes that the functions all have similar amounts of 'curviness'. However as the number of zeros of the vector fields acting on the functions increases the 'curviness' grows, such as for example when the system being studied is more complicated (has more zeros), the number of terms needed will increase. So as we move further out along the chain of figure 9 (p. 10) each function will require more terms for an approximation as accurate as those in the previous systems.

3.3 The PSRP Limit Space

It would obviously be ideal to be able to describe the limit space of the PSRP recursion. Unfortunately at this time that has not been worked out. The monograph *Self-Similar Groups*[3] provides the best current lead in that direction. We have mentioned the intimate connection between self-similar groups and automata already, however the work [3] provides a much more general formalism for these groups, called the iterated monodromy group of a partial self-covering of topological spaces. It seems clear that a 'partial self-covering' is precisely the kind of object that is going to be capable of solving the PSRP. The most general formalism for describing such a covering, from a talk [7] given by the author of [3] requires a finite covering map $p : \mathcal{M}_1 \to \mathcal{M}$, and a continuous map $\iota : \mathcal{M}_1 \to \mathcal{M}$, these in turn induce maps $p_* : \Pi(\mathcal{M}_1) \to \Pi(\mathcal{M})$, and $\iota_* : \Pi(\mathcal{M}_1) \to \Pi(\mathcal{M})$. The result is that ι_* is a virtual endomorphism of $\Pi(\mathcal{M}_1)$, a self-similar group, and the iterated monodromy of the (topological) correspondence between $\mathcal{M}_1$ and $\mathcal{M}$, furthermore the fundamental group $\Pi(\mathcal{M}_1)$ is a sub-group of $\Pi(\mathcal{M})$ through the isomorphism p_*.

Unfortunately, the maps we have so far described are neither continuous nor a covering, and in fact for the dynamical systems $\mathcal{D}_i$ (which is what we are really interested in) we really only have the one map p_{D_i} for each pair. One potential solution is to change the 'interpretational schema' of the syntax of the models M_i. (More properly we should say change the model space of the syntax, but that label seems likely to add significant confusion overlapping as it does with the use of the word model that we have already employed.) If we thought of the symbols of each model as amplitudes for a set of basis functions, such as for example polynomials or exponentials, then we would naturally have a continuous function defined on the systems $\mathcal{D}_i$, but we would still need to find a relationship between those functions across different systems $\mathcal{D}_i$. There are reasons to believe

that a more thourough understanding of the material in [3] may yet yield the needed maps. Finally a more direct approach using the profinite completion of the fundamental groupoids of the systems $\mathcal{D}_i$ also seems quite promising.

References

[1] Blum, L., Cucker, F., Shub, M., Smale, S., *Complexity and Real Computation*, Springer-Verlag, New York, 1998

[2] Eilenberg, S., *Automata, Languages, and Machines Vol A*, Academic Press, New York, 1974

[3] Nekrashevych, V., *Self-Similar Groups*, Mathematical Surveys and Monographs, the American Mathematical Society, U.S.A., 2005

[4] Balcar, B., and Franek, F., Structural Properties of Universal Mimimal Dynamical Systems for Dicrete Semigroups, Transactions of the American Mathematical Society, Vol. 349, No. 5, May 1997, pp. 1697-1724

[5] Pour-El, M.B., Richards J.I., *Computability in Analysis and Physics* Springer-Verlag, 1989

[6] Patterson, A.L.T., *Groupoids, Inverse Semi-Groups, and their Operator Algebras* Birkhauser, Boston, 1998

[7] Nekrashevych, V., *Iterated Monodromy Groups* delived at Bath, 2009